Hybrid RSA + AES File Encryption System

Gurvir Singh

October 2024

1 Abstract

This project presents a secure and efficient hybrid encryption system combining RSA and AES encryption algorithms. The system leverages RSA for encrypting a randomly generated AES key and AES for encrypting files of any type, including text files and binary files. RSA, an asymmetric encryption technique, provides a secure method for exchanging the AES key, while AES, a symmetric encryption algorithm, ensures fast and efficient encryption of large file data. The project implements RSA with OAEP padding, enhancing security against modern cryptographic attacks. A Python-based implementation is provided, using the `cryptography` and `sympy` libraries for encryption, decryption, and key management. The system is tested on various file sizes to validate performance and security. Future improvements, such as adding digital signatures for authentication, performance benchmarking for larger files, and a user-friendly interface, are also discussed.

2 Introduction

The goal of this project is to develop a secure and efficient encryption system using a hybrid approach that combines the strengths of both RSA (Rivest-Shamir-Adleman) and AES (Advanced Encryption Standard) algorithms. In cryptography, one of the main challenges is finding a balance between strong encryption (security) and performance (speed). RSA, a well-known asymmetric algorithm, provides strong encryption for small data, but it is computationally expensive for larger files. AES, on the other hand, is a symmetric encryption method that is fast and efficient for large data. However, it relies on secure key distribution.

This project addresses these issues by encrypting files with AES, and then securing the AES key with RSA, which allows for secure key exchange between parties. The combination of these two algorithms ensures both security and performance, making it ideal for protecting large files while maintaining strong encryption.

3 Theory

RSA Encryption RSA is an asymmetric encryption algorithm, meaning it uses two different keys: a public key for encryption and a private key for decryption. It is widely used in secure communications because of its ability to encrypt data in a way that only the owner of the private key can decrypt.

The core mathematical concepts behind RSA rely on the difficulty of factoring the product of two large prime numbers, a problem that becomes increasingly difficult as the key size increases. The RSA algorithm consists of three main steps:

1. **Key Generation**:
   - Two large prime numbers, p and q, are generated.
   - The product of these primes, $n = p \times q$, is calculated.
   - The public exponent e is chosen (often 65537 for efficiency and security).
   - The private exponent d is calculated such that $d \times e \equiv 1 \pmod{\phi(n)}$, where $\phi(n)$ is Euler's totient function, $(p-1)(q-1)$.
2. **Encryption**: A message is converted into a number m and encrypted using the public key as $c = m^e \mod n$, where c is the ciphertext.
3. **Decryption**: The ciphertext is decrypted using the private key as $m = c^d \mod n$, recovering the original message.

RSA and Key Exchange RSA is primarily used for secure key exchange in hybrid encryption systems. In this project, it is used to encrypt the AES key, ensuring that only the holder of the RSA private key can decrypt and obtain the AES key. The encrypted AES key is much smaller than the file content, which makes RSA suitable for this purpose despite its slower performance on large data.

AES Encryption AES (Advanced Encryption Standard) is a symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. AES is widely regarded as one of the most secure encryption methods and is used globally for protecting sensitive information.

The AES algorithm operates on blocks of data (typically 128 bits) and can use key sizes of 128, 192, or 256 bits. It transforms the plaintext through multiple rounds of substitution, permutation, and mixing operations, making it highly secure against various cryptographic attacks. AES in CFB mode (Cipher Feedback) allows for encryption of data of any size and removes the need for padding, making it more flexible for different file types.

Key Features of AES:

1. **Speed**: AES is highly efficient for large files and data streams.
2. **Security**: With a 256-bit key, AES is considered unbreakable by brute force attacks.
3. **Flexibility**: AES in CFB mode can encrypt data of any length, making it ideal for both text files and binary data.

Hybrid Approach: RSA + AES A hybrid encryption system leverages the strengths of both RSA and AES:

- RSA is used to encrypt the AES key because of its secure key exchange properties.
- AES is used to encrypt the actual file data, due to its high performance when dealing with large amounts of data.

By combining RSA and AES, the system achieves the best of both worlds: secure key exchange through RSA and efficient file encryption through AES. The AES key is generated randomly for each encryption session, and the RSA public key encrypts this key. The encrypted AES key is then included with the encrypted file, ensuring that only the holder of the RSA private key can decrypt the file by first decrypting the AES key.

Why Hybrid Encryption?

- **Security**: The RSA encryption of the AES key ensures that the key is securely transferred between the sender and recipient, even if the communication channel is compromised.
- **Performance**: AES can handle large data efficiently, which would be impractical for RSA. By using AES to encrypt the data and RSA to encrypt the key, we get the security of RSA with the speed of AES.

4 Implementation

4.1 Libraries Used

This project makes use of the following Python libraries:

- `cryptography`: Provides the necessary cryptographic primitives to implement RSA and AES encryption algorithms.
- `sympy`: Used for generating prime numbers and performing mathematical computations necessary for RSA.

To install these dependencies:

```
pip install cryptography sympy
```

4.2 Key Functions

RSA Key Generation (`generate_rsa_keys`) This function generates a 2048-bit RSA key pair, consisting of a public and private key, used for encrypting and decrypting the AES key.

```
def generate_rsa_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return public_key, private_key
```

AES Encryption (`aes_encrypt`) This function encrypts the file content using AES encryption in **CFB mode** (Cipher Feedback Mode). It takes the data and an AES key as input and returns the AES-encrypted data.

```
def aes_encrypt(data, aes_key):
    iv = os.urandom(16)  # Generate a 16-byte IV
    cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(data) + encryptor.finalize()  # Encrypt the data
    return iv + ciphertext  # Prepend the IV to the ciphertext
```

AES Decryption (`aes_decrypt`) This function decrypts AES-encrypted file data using the AES key and the initialization vector (IV) from the encrypted data.

```
def aes_decrypt(ciphertext, aes_key):
    iv = ciphertext[:16]  # Extract the first 16 bytes as the IV
    cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    return decryptor.update(ciphertext[16:]) + decryptor.finalize()  # Decrypt the file data
```

5 Conclusion

In this project, we successfully designed and implemented a hybrid encryption system that combines the strengths of both RSA and AES algorithms. By using RSA for secure key exchange and AES for fast, efficient file encryption, we achieved a system that provides both security and performance. The use of 2048-bit RSA keys ensures strong protection for the AES key, while AES-256 encryption provides robust encryption for large files.

The project demonstrates the effectiveness of hybrid encryption, particularly in environments where both security and performance are critical. The

RSA algorithm is well-suited for securely exchanging the AES key, even over potentially insecure channels, while AES offers efficient and fast encryption for large data, reducing the performance overhead that would occur if RSA were used for file encryption alone.

Testing confirmed that the system performs well with files of various sizes, showing minimal overhead during encryption and decryption processes. Additionally, the system adheres to modern cryptographic standards, ensuring protection against brute-force attacks and other vulnerabilities. The use of OAEP padding in RSA further enhances security by mitigating risks of certain cryptographic attacks.

Despite its success, there are areas for future improvement. Potential enhancements include the implementation of digital signatures to verify file authenticity, making the system resistant to tampering. Further performance optimizations, particularly when handling very large files, could also be explored. Additionally, developing a graphical user interface (GUI) would make the system more user-friendly, allowing non-technical users to easily encrypt and decrypt files.

Overall, this project highlights the benefits of hybrid encryption in real-world applications where both performance and security are required. By leveraging the advantages of RSA and AES, this system provides a robust solution for secure file encryption and key management.